Model-Based Methods in Reinforcement Learning: From Markov Decision Processes to Monte Carlo Tree Search

Ahmed BADI
ahmedbadi905@gmail.com
linkedin.com/in/badi-ahmed

January 2026

Abstract

Model-based methods in reinforcement learning (RL) rely on an explicit model of the environment to predict future states and rewards. Instead of learning behavior only by trial and error, a model-based agent can imagine or simulate different action sequences and choose the best one before acting. This article gives a clear and intuitive introduction to model-based RL. We start with Markov Decision Processes (MDPs) and the Bellman equation, then present dynamic programming algorithms such as value iteration and policy iteration. Next, we discuss planning techniques like Monte Carlo Tree Search (MCTS), which simulate trajectories to guide decisions in large state spaces. Throughout, we include mathematical formulations, examples, and comparisons between model-based and model-free methods in terms of sample efficiency, computation, and robustness.

Keywords: Model-Based Reinforcement Learning, Markov Decision Process, Bellman Equation, Value Iteration, Policy Iteration, Monte Carlo Tree Search, Planning.

1 Introduction

Reinforcement Learning (RL) is about learning how to act by interacting with an environment and receiving rewards over time. In many RL problems, the agent does not know how the environment behaves and must discover good behavior by trial and error. [1], [2]

Model-based methods add an extra layer: they try to learn or exploit a *model* of the environment. This model predicts what happens next when the agent takes an action in a given state. Once the agent has such a model, it can simulate many possible futures and plan a sequence of actions without physically trying all of them. [3], [4]

1.1 What is a Model in RL?

In the RL context, a model usually refers to:

- The transition dynamics $P(s' \mid s, a)$: the probability of reaching next state s' from state s after action a.
- The reward function $R(s, a, s')$: the expected reward for that transition.

A model-based agent uses this information to **predict outcomes** and choose actions by planning, instead of relying only on trial-and-error updates. [1], [5]

1.2 Model-Based vs Model-Free

- **Model-free methods** (like Q-learning) learn value functions or policies directly from experience, without explicitly modeling the environment's dynamics.
- **Model-based methods** learn or use a model and then apply planning algorithms (such as dynamic programming or tree search) on top of that model.

Model-based RL can be:

- More **sample efficient**, because each real interaction can be reused in many simulated rollouts.
- More **computationally expensive**, because planning over models requires extra computation.

[3], [4]

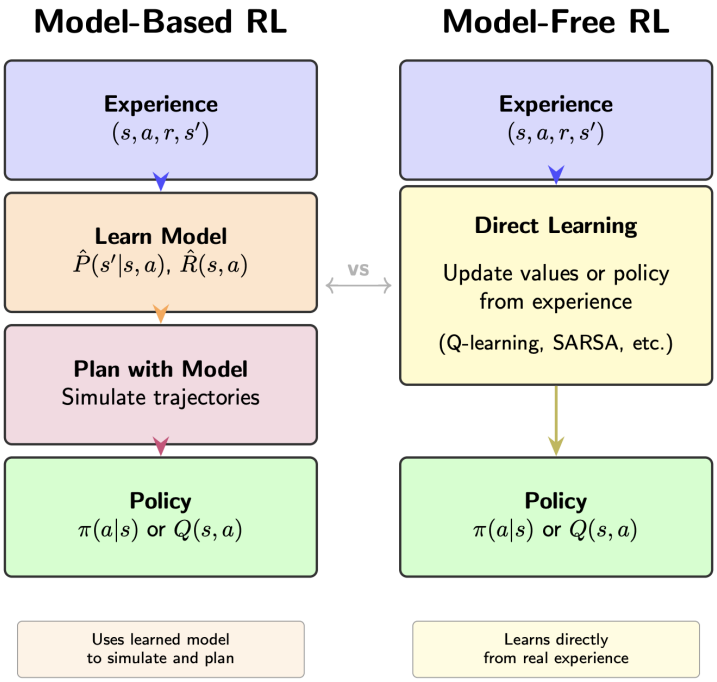

Figure 1: High-level comparison: model-based RL learns or uses a model of transitions and rewards to plan; model-free RL directly learns value functions or policies from data.

2 Markov Decision Processes

Model-based RL is usually built on the formalism of Markov Decision Processes (MDPs), which describe decision-making in stochastic environments. [1], [5]

2.1 Definition of an MDP

An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$: set of states.
- $\mathcal{A}$: set of actions.
- $P(s' \mid s, a)$: transition probability from s to s' under action a.

- $R(s, a, s')$: expected reward for transitioning from s to s' using action a.
- $\gamma \in [0, 1)$: discount factor for future rewards.

[5], [6]

2.2 Markov Property

The Markov property means that the future depends only on the current state and action, not on the entire history:

$$P(S_{t+1} = s' \mid S_0, A_0, \ldots, S_t = s, A_t = a) = P(s' \mid s, a).$$

This allows compact modeling and makes dynamic programming feasible. [1]

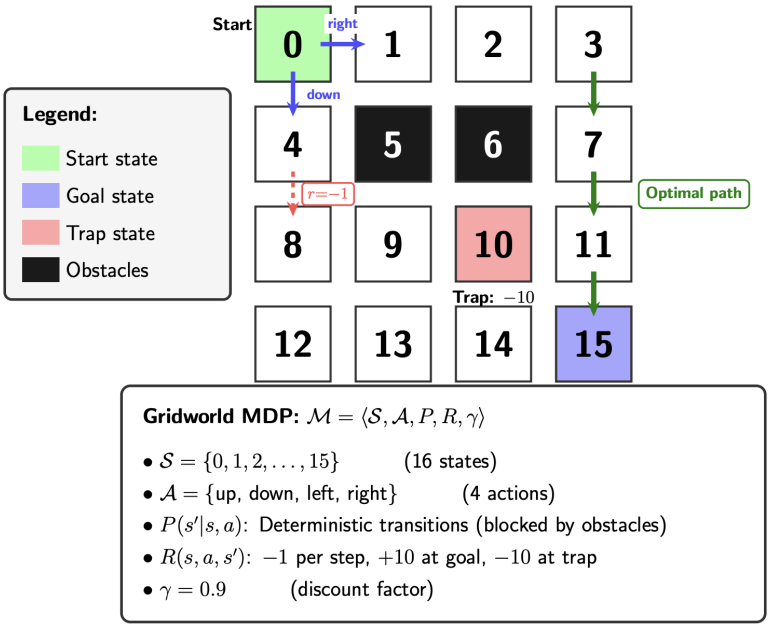

Figure 2: Example MDP: a small gridworld where the agent moves in four directions and receives rewards when reaching terminal states.

2.3 Policies and Return

A policy $\pi(a \mid s)$ gives a probability distribution over actions in each state. The agent's goal is to find a policy $\pi^\star$ that maximizes expected *return*, defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}.$$

Model-based methods use the MDP model P and R together with policies to compute or approximate optimal returns. [1]

3 Value Functions and the Bellman Equation

Value functions quantify how good states or state-action pairs are, given a policy. The Bellman equations are central to model-based RL because they connect value functions to the environment model. [1], [7]

3.1 State-Value and Action-Value Functions

Given a policy π:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right],$$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right].$$

These expectations are defined with respect to the transition probabilities P and rewards R, which are part of the model. [1]

3.2 Bellman Expectation Equations

The Bellman equation for v_π expresses the value of a state in terms of the model and the values of successor states:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right].$$

Similarly, for q_π:

$$q_\pi(s, a) = \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right].$$

[1], [8]

3.3 Bellman Optimality Equations

The optimal value function $v_\star(s) = \max_\pi v_\pi(s)$ satisfies

$$v_\star(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\star(s') \right],$$

and the optimal action-value function $q_\star(s, a)$ satisfies

$$q_\star(s, a) = \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma \max_{a'} q_\star(s', a') \right].$$

These equations use the model to describe optimal behavior and are the basis for planning algorithms like value iteration. [1], [7]

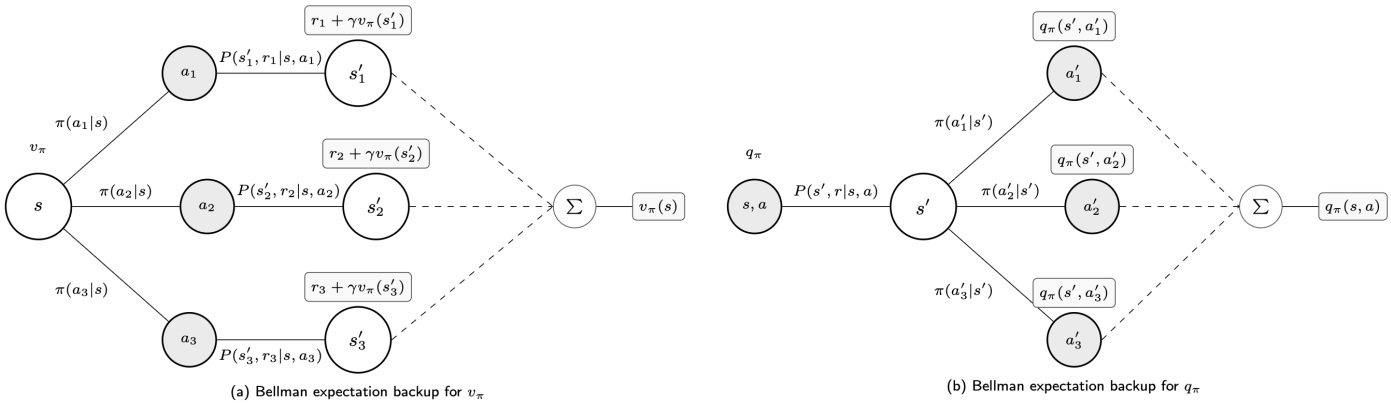

Figure 3: Graphical view of a Bellman backup: value of a state is updated from immediate reward and values of successor states under the model.

4 Dynamic Programming for Planning

Dynamic programming (DP) methods compute optimal value functions and policies assuming the model P and R is known. They are classic examples of model-based methods. [1], [8]

4.1 Policy Evaluation

For a fixed policy π, policy evaluation iteratively applies the Bellman expectation equation:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right].$$

This iteration converges to v_π under mild conditions. [1]

4.2 Policy Improvement and Policy Iteration

Given v_π, we can improve the policy by acting greedily:

$$\pi_{\text{new}}(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right].$$

Policy iteration alternates policy evaluation and policy improvement until convergence to an optimal policy. [1], [8]

4.3 Value Iteration

Value iteration combines evaluation and improvement in a single update:

$$v_{k+1}(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right].$$

This repeatedly applies the Bellman optimality operator until v_k is close to $v_\star$. The optimal policy is then

$$\pi^\star(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_\star(s') \right].$$

[1], [9]

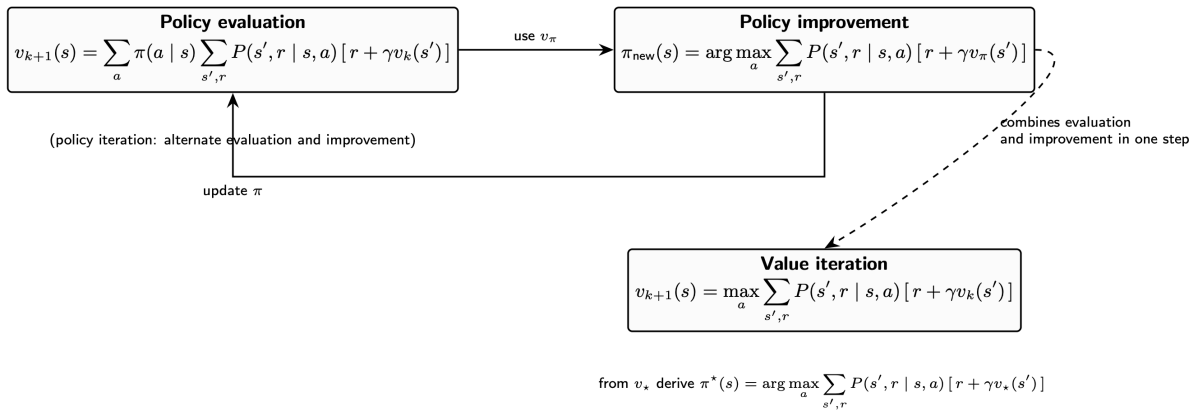

Figure 4: Visualization of the process of iteration to attend convergence

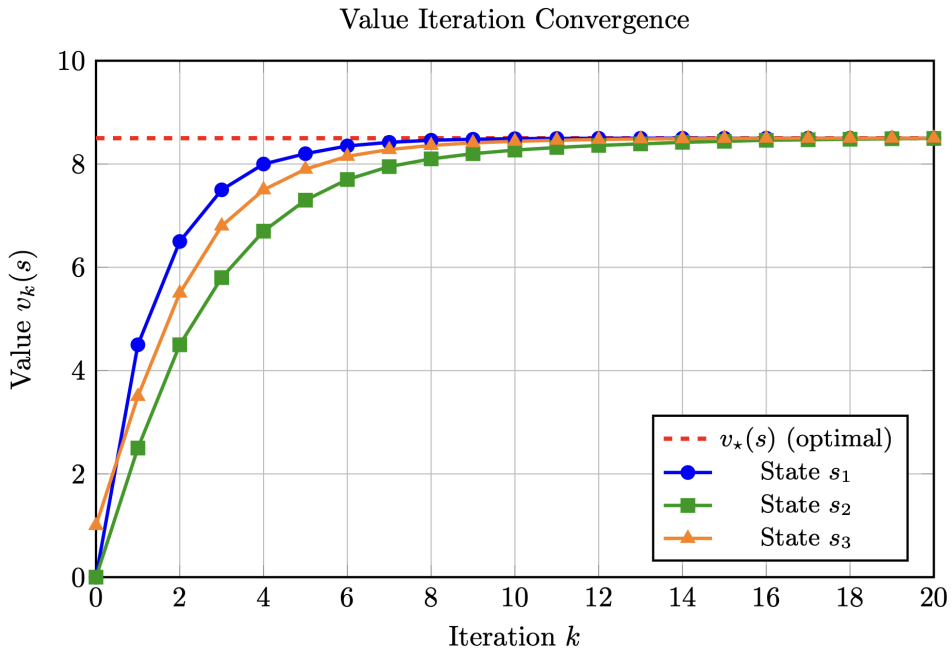

Figure 5: Example: value iteration gradually converging to the optimal value function over iterations.

4.4 Policy Iteration vs Value Iteration

Table 1 summarizes policy iteration and value iteration.

Method	Core idea	Pros	Cons
Policy iteration	Alternate evaluating and improving a policy	Often converges in few iterations	Each evaluation step may be expensive
Value iteration	Directly update values via Bellman optimality	Simple update, easy to implement	Many iterations may be needed for tight convergence

Table 1: Comparison of policy iteration and value iteration in dynamic programming.

5 Model-Based RL with Learned Models

In many real problems, we do not know P and R exactly. Model-based RL then tries to *learn* a model from experience and uses DP-style planning on the learned model. [3], [4]

5.1 Learning the Model

The agent can estimate:

- Transition probabilities $\hat{P}(s' \mid s, a)$ from observed counts or parametric models.
- Rewards $\hat{R}(s, a, s')$ as averages of observed rewards.

For finite MDPs, simple frequency counts can be used:

$$\hat{P}(s' \mid s, a) = \frac{N(s, a, s')}{\sum_{s''} N(s, a, s'')},$$

where $N(s, a, s')$ is the number of times transition $s \xrightarrow{a} s'$ was observed. [1]

5.2 Dyna-style Algorithms

Dyna is a classic framework that combines model-free learning with model-based planning: [1]

- **Direct RL**: update value functions from real experience.
- **Model learning**: update $\hat{P}$ and $\hat{R}$ from experience.
- **Planning**: simulate transitions using the learned model and apply extra updates.

This makes each real experience more valuable by generating many simulated experiences from the model.

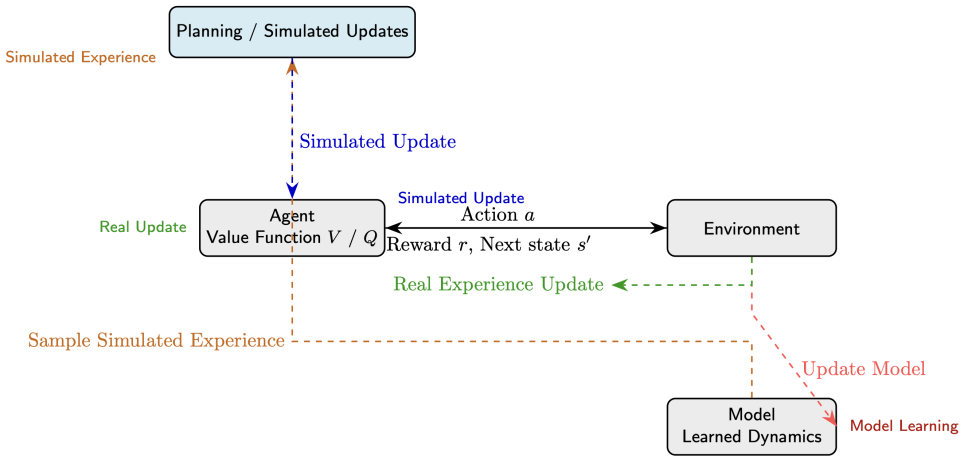

Figure 6: Sketch of a Dyna-style architecture: real experience updates both the model and the value function; planning uses the model to generate simulated updates.

6 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search is a powerful planning technique that builds a search tree by simulating many trajectories from the current state. It is widely used in game-playing systems like AlphaGo and in RL settings with large state spaces. [10], [11]

6.1 Basic Idea

MCTS incrementally grows a search tree by repeating four steps:

1. **Selection**: starting from the root, select child nodes according to a tree policy (often based on UCB).
2. **Expansion**: add one or more new child nodes when reaching a leaf.
3. **Simulation (Rollout)**: simulate a random or heuristic policy until a terminal state.
4. **Backpropagation**: propagate the obtained return back up the tree, updating value estimates.

[10], [11]

6.2 Using a Model in MCTS

MCTS relies on a model (or simulator) to generate next states and rewards during simulation:

- In game-playing, the rules of the game give an exact model.
- In RL, a learned model can be used to simulate possible futures from the current state.

This makes MCTS an **online planning** method: it searches from the current state each time before taking an action. [11], [12]

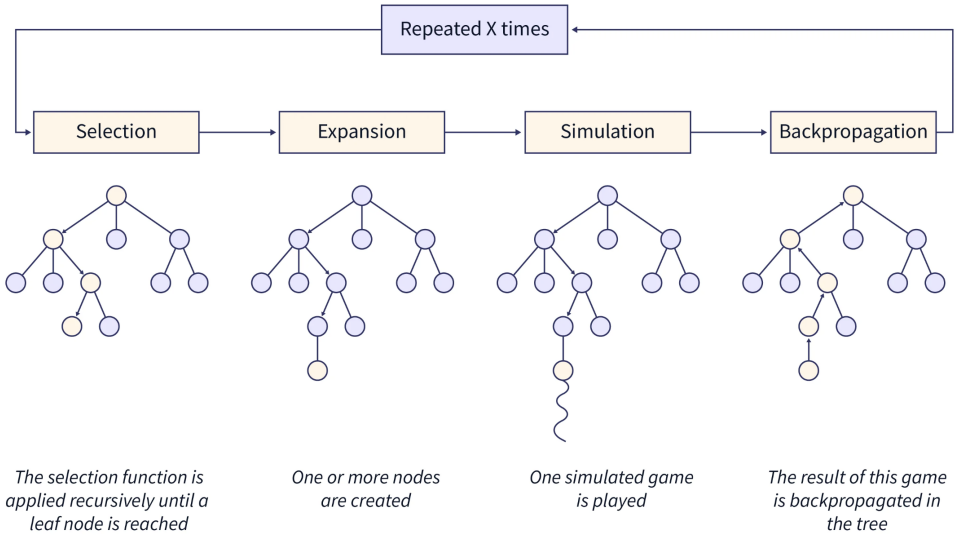

Figure 7: Monte Carlo Tree Search: repeated selection, expansion, simulation, and backpropagation grow a search tree around the current state.

Source: `https://www.scaler.com/topics/images/outline-of-monte-carlo-tree-search.webp`

6.3 Upper Confidence Bounds in MCTS

A popular selection rule in MCTS is UCT (Upper Confidence bounds applied to Trees), which balances exploitation and exploration:

$$a^\star = \arg\max_a \left[\hat{Q}(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right],$$

where:

- $\hat{Q}(s,a)$ is the estimated value of action a in state s.
- $N(s)$ is the visit count of state s.
- $N(s,a)$ is the visit count of edge (s,a).
- C controls exploration strength.

[10], [12]

7 Examples and Simple Statistics

Model-based methods often show different sample and computation profiles compared to model-free methods. [3], [4]

7.1 Sample Efficiency vs Computation

- Model-based RL tends to reach good performance with fewer real environment interactions, because each interaction feeds the model and planning can reuse this information many times.
- However, planning (value iteration, MCTS) can be computationally heavy per decision.

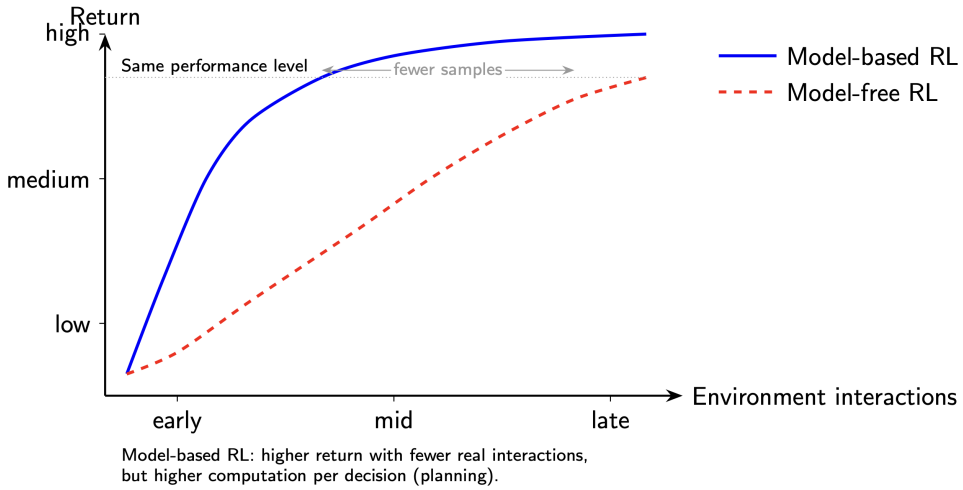

Figure 8: Illustrative learning curves: model-based RL achieving higher return with fewer environment steps than a model-free baseline, at the cost of more computation per step.

7.2 Gridworld Example

In a small gridworld:

- Dynamic programming with a known model converges to the optimal policy in a predictable number of value iteration steps.
- Q-learning, using only experience, may require many more episodes to discover the same policy.

[1], [9]

8 Advantages and Limitations of Model-Based Methods

8.1 Advantages

- **Sample efficiency**: Less real-world interaction needed to learn good policies. [3], [4]
- **Flexibility**: Once a model is available, it can be reused for different reward functions or goals.
- **Planning and imagination**: The agent can test risky strategies in simulation instead of the real system.

8.2 Limitations

- **Model bias**: If the learned model is inaccurate, planning may suggest poor actions. [4]
- **Computational cost**: Planning over large state spaces (e.g., big MDPs, deep trees) can be expensive.
- **Complexity in high dimensions**: Learning and using accurate models in complex environments (images, continuous control) is hard.

9 Connections to Modern Deep RL

Modern deep RL systems often combine model-free learning with model-based planning techniques such as MCTS. For example, AlphaGo and AlphaZero used neural networks as value and policy approximators together with MCTS to search game trees. [11]

9.1 Model-Based Deep RL

Recent research explores:

- Learning latent dynamics models with neural networks.
- Planning in latent spaces (e.g., using imagination rollouts).
- Combining model-based rollouts with model-free value updates.

[4]

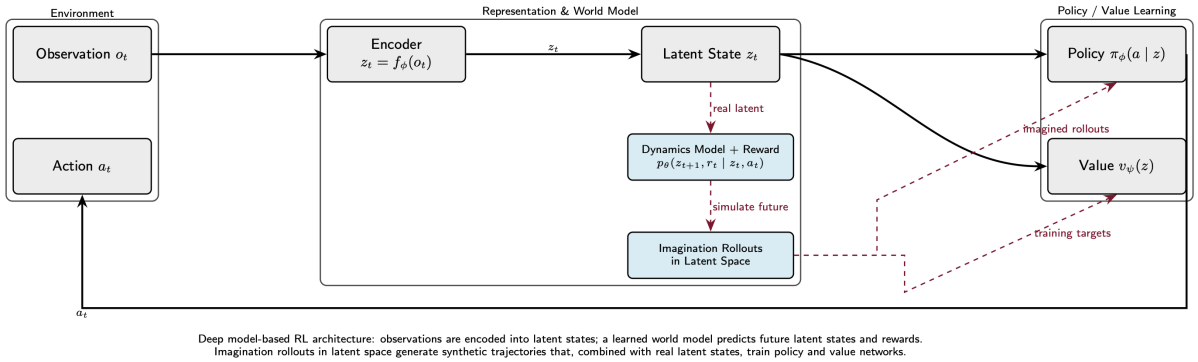

Figure 9: Example deep model-based RL architecture: a neural model predicts next latent states and rewards; planning or rollouts use this model to improve policies and value functions.

10 Conclusion

Model-based methods in reinforcement learning enrich the basic RL framework with predictive models of the environment. These models make it possible to:

- Apply dynamic programming algorithms like value iteration and policy iteration.
- Use powerful planning tools such as Monte Carlo Tree Search.
- Increase sample efficiency by reusing experience in simulated rollouts.

However, model-based RL trades sample efficiency for computational cost and introduces the risk of model bias if the learned model is inaccurate. For many real-world problems, combining model-based and model-free ideas is a promising way to balance these trade-offs. [3], [4]

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] Anonymous, *Introduction to reinforcement learning*, 2024. [Online]. Available: https://arxiv.org/html/2408.07712v2

[3] ScienceDirect Topics, *Model-based reinforcement learning – an overview*, 2021. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/model-based-reinforcement-learning

[4] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *Foundations and Trends in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023. [Online]. Available: https://www.nowpublishers.com/article/DownloadSummary/MAL-086

[5] DeepAI, *Markov decision process definition*, 2019. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/markov-decision-process

[6] Wikipedia contributors, *Markov decision process*, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Markov_decision_process

[7] Hugging Face Deep RL Course, *The bellman equation: Simplify our value estimation*, 2024. [Online]. Available: https://huggingface.co/learn/deep-rl-course/en/unit2/bellman-equation

[8] S. Yahyah, *Dynamic programming for reinforcement learning, the bellman equation*, 2023. [Online]. Available: https://suzyahyah.github.io/reinforcement%20learning/2023/02/01/DP-RL-Bellman.html

[9] StudySmarter, *Value iteration: Algorithm & examples*, 2024. [Online]. Available: https://www.studysmarter.co.uk/explanations/engineering/artificial-intelligence-engineering/value-iteration/

[10] GeeksforGeeks, *Monte carlo tree search (mcts) in machine learning*, 2021. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/monte-carlo-tree-search-mcts-in-machine-learning/

[11] M. Fujita, G. Tesauro, M. Bowling, et al., "On monte carlo tree search and reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 1–40, 2022. [Online]. Available: https://jair.org/index.php/jair/article/view/11099

[12] C. B. Browne et al., "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.